U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

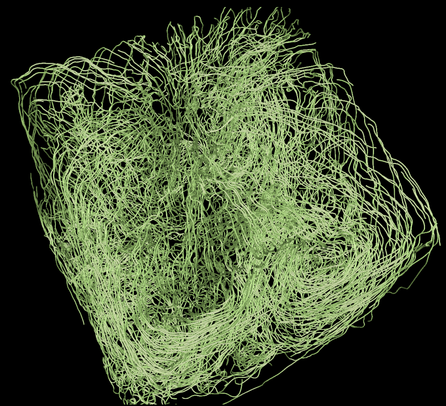
Distributed Data Analysis at Scale

"Data movement, rather than computational processing, will be the constrained resource at exascale." – Dongarra et al. 2011.

Tom Peterka
tpeterka@mcs.anl.gov
http://www.mcs.anl.gov/~tpeterka
Mathematics and Computer Science Division

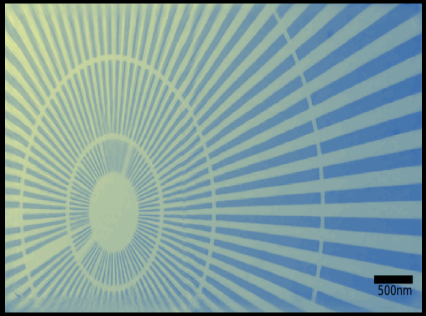
EDF-INRIA Seminar
June 24, 2016

Examples

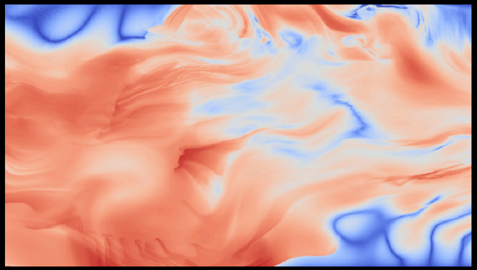


Streamlines and pathlines
in nuclear engineering

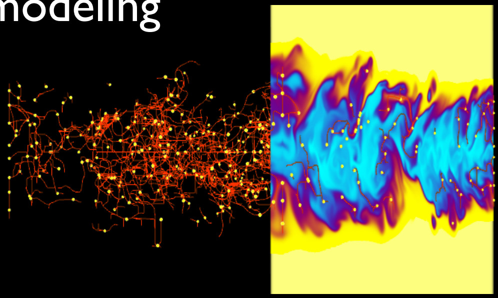
Stream surfaces
in meteorology



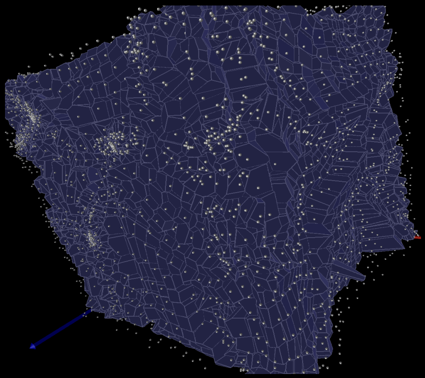
Ptychography
in materials science



FTLE
in climate modeling



Morse-Smale complex
in combustion



Voronoi and Delaunay tessellation
in cosmology

Communication Design Patterns

Analysis	Application	Application Data Model	Analysis Data Model	Analysis Algorithm	Communication	Additional
Particle Tracing	CFD	Unstructured Mesh	Particles	Numerical Integration	Nearest neighbor	File I/O, Domain decomposition, process assignment, utilities
Information Entropy	Astrophysics	AMR	Histograms	Convolution	Global reduction, nearest neighbor	
Morse-Smale Complex	Combustion	Structured Grid	Complexes	Graph Simplification	Global reduction	
Computational Geometry	Cosmology	Particles	Tessellations	Voronoi	Nearest neighbor	

You do this yourself

Can use serial libraries such as OSUFlow, Qhull, VTK (don't have to start from scratch)

DIY handles this

Keys:

- Separate custom application code from reusable communication
- Recognize that diverse applications use a common set of design patterns.

A Data Movement Library for HPC Data Analysis

Ad hoc

Application	
Analysis Algorithm	
Merge-reduce	Swap-reduce
Iterative	Nearest Neighbor
OS / Runtime	

Structured

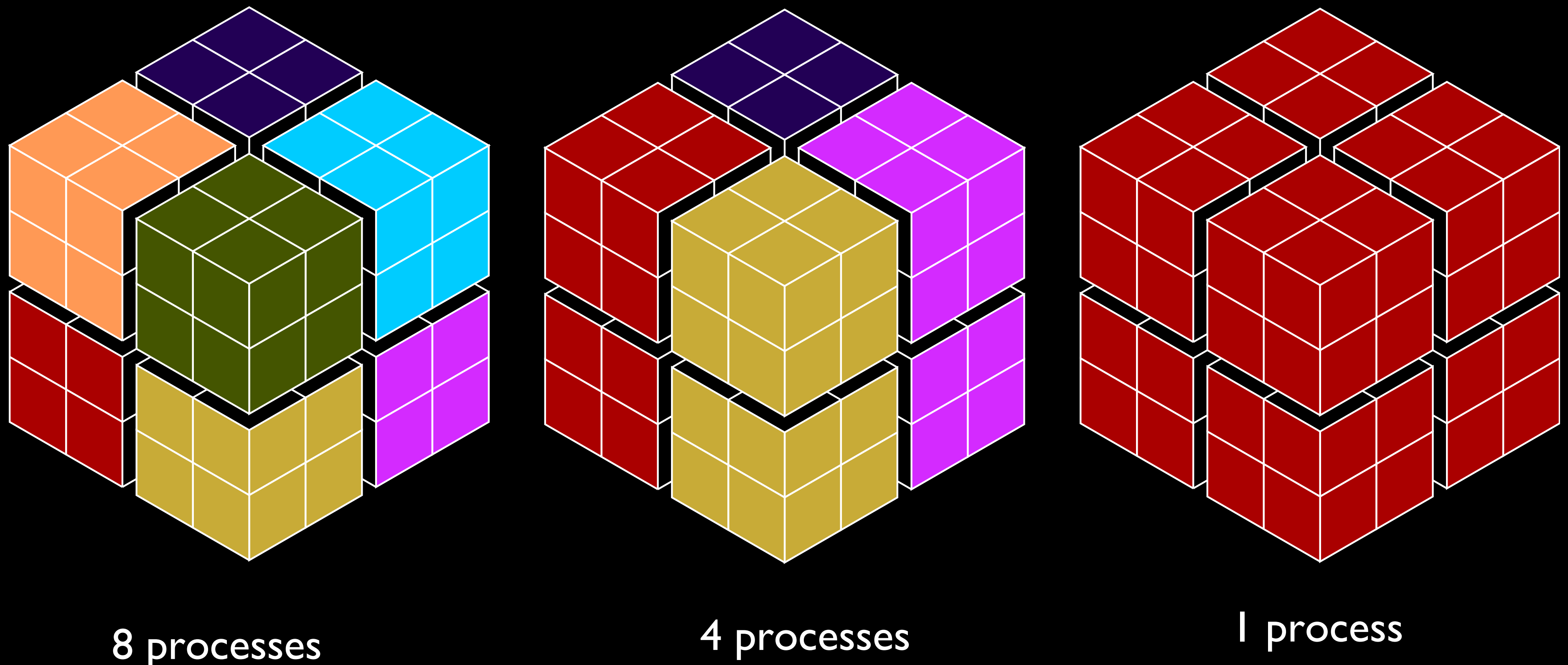
- Application
- Analysis Algorithm — OSUFlow, Tess2, Cian2, Gaia, ITL, VTK
- Data Movement — DIY
- OS / Runtime

```
void ParallelAlgorithm() {
  …
  MPI_Send();
  …
  MPI_Recv();
  …
  MPI_Barrier();
  …
  MPI_File_write();
}
```

```
void ParallelAlgorithm() {
  …
  foreach(&LocalAlgorithm);
  exchange();
  reduce();
  write_blocks();
}
void LocalAlgorithm() {
  …
}
```

Basic Concepts

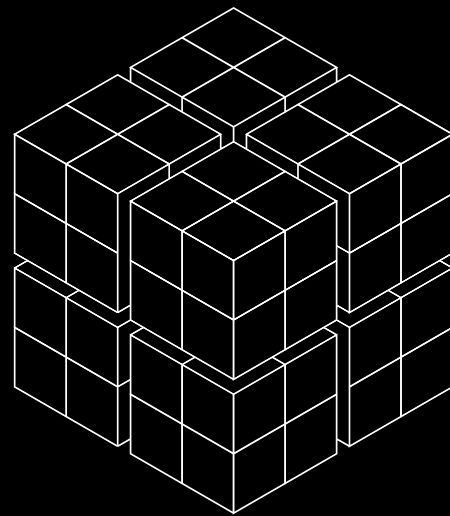
Block Parallelism



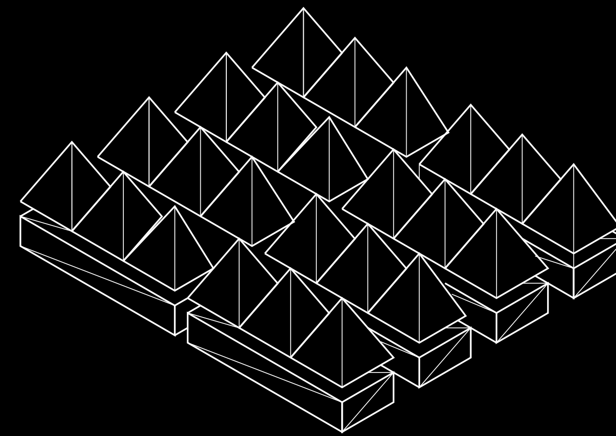
Blocks are units of work and communication; blocks exchange information with each other using DIY's communication algorithms. DIY manages block placement in MPI processes and memory/storage. This allows for flexible, high performance programs that are easy to write and debug.

Partition Data Into Blocks

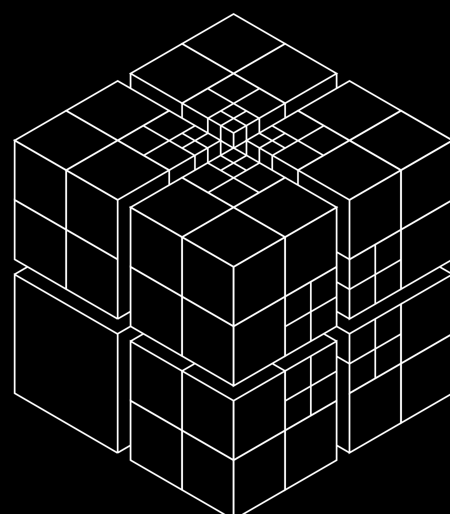
The block is the basic unit of data decomposition. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. Blocks don't have to be "blocky." Any subdivision of data (eg., a set of graph nodes, a group of particles, etc.) is a block.



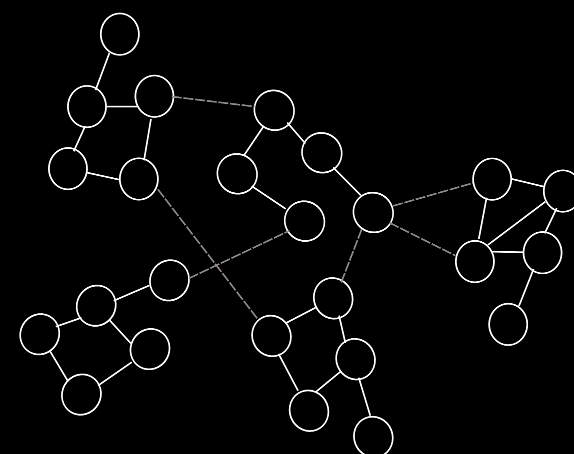
Structured Grid



Unstructured Mesh



AMR Grid



Graph

Multiple Regular Decompositions

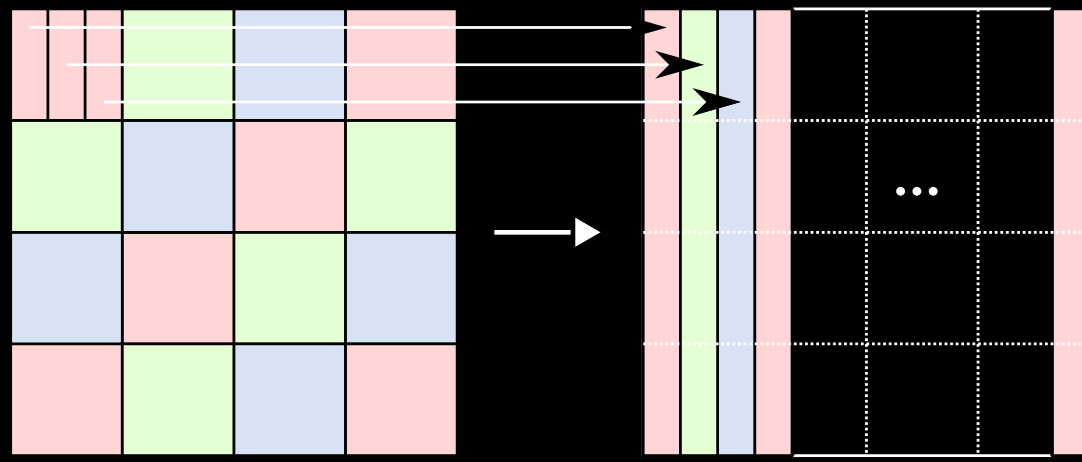
1. Decomposition can be a regular grid of blocks or a k-d tree.

2. For a regular grid, constraints on numbers of blocks can be imposed to get pencil or slab shapes.

3. Multiple decompositions can co-exist.

Original block decomposition

Slab or pencil decomposition for FFT

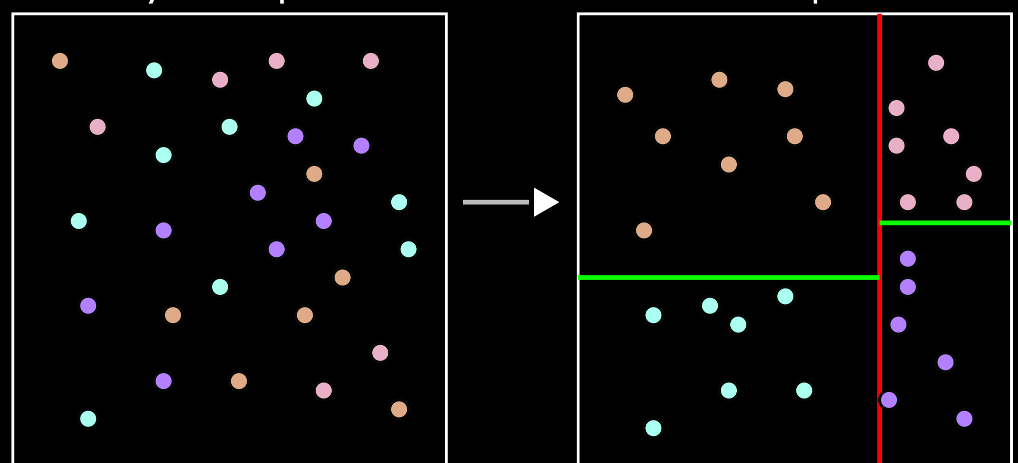


16 blocks, 3 procs indicated by color

Need not be same number of blocks

Original data
Arbitrary decomposition

Kd-tree decomposition

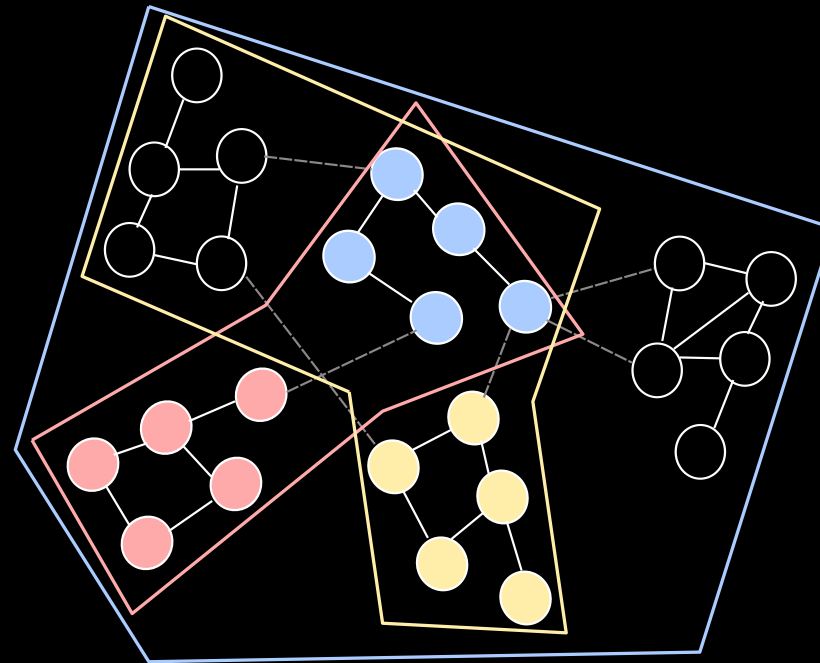


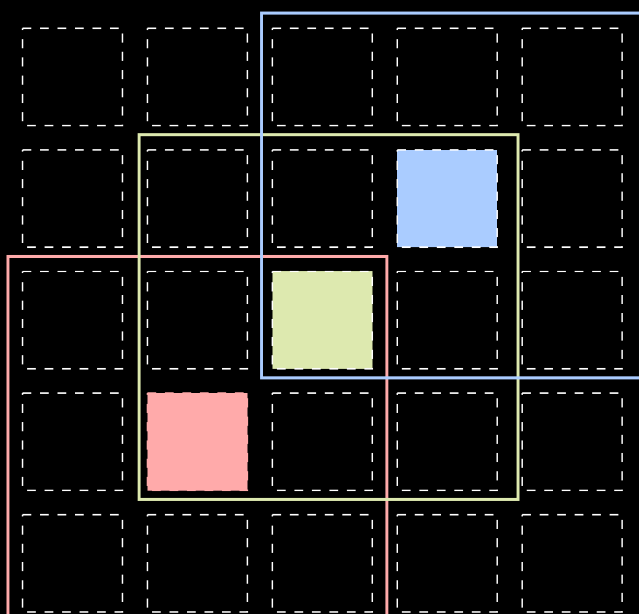
4 blocks indicated by color.
No spatial locality assumed.

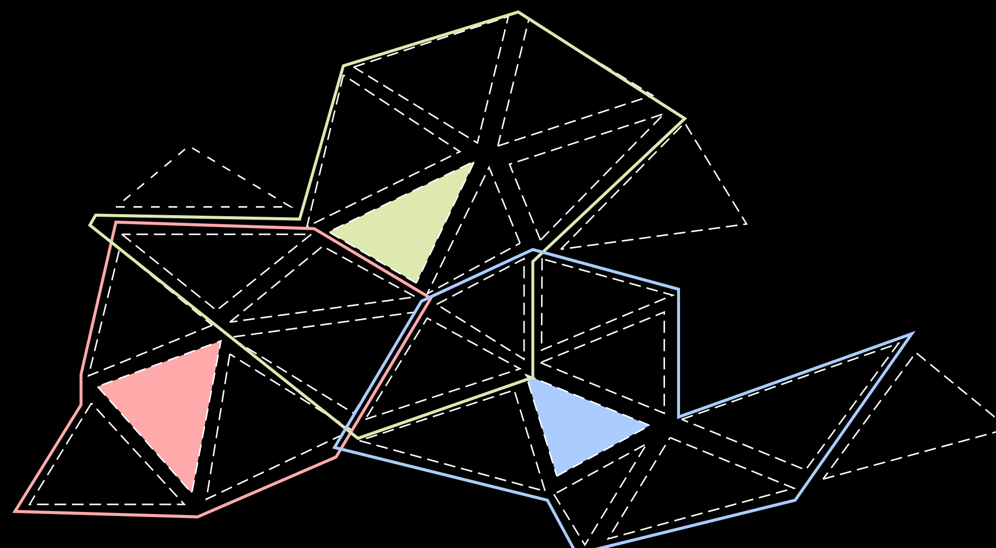
4 new blocks spatially contiguous and load balanced by number of objects in each.

Neighborhood Links

- Limited-range communication
- Allow arbitrary groupings
- Distributed, local data structure and knowledge of other blocks (not master-slave global knowledge)







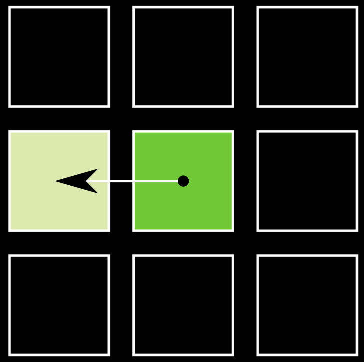
Examples of 3 neighborhoods in a regular grid, unstructured mesh, and graph.

Different Neighborhood Communication Patterns

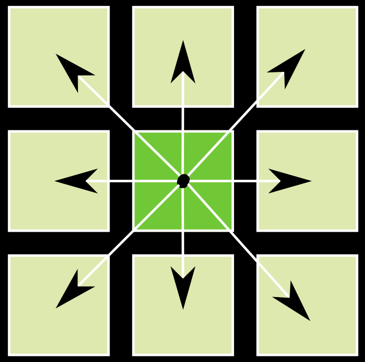
DIY provides point to point and different varieties of collectives within a neighborhood via its enqueue/exchange/dequeue mechanism.

How to enqueue items for neighbor exchange

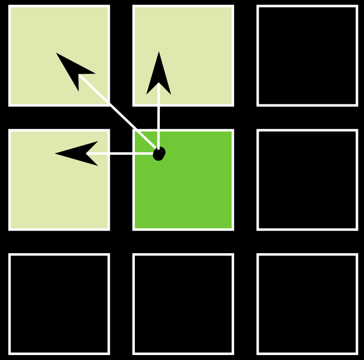
- DIY offers several options
- Send to a particular neighbor or neighbors, send to all nearby neighbors, send to all neighbors
- Support for periodic boundary conditions



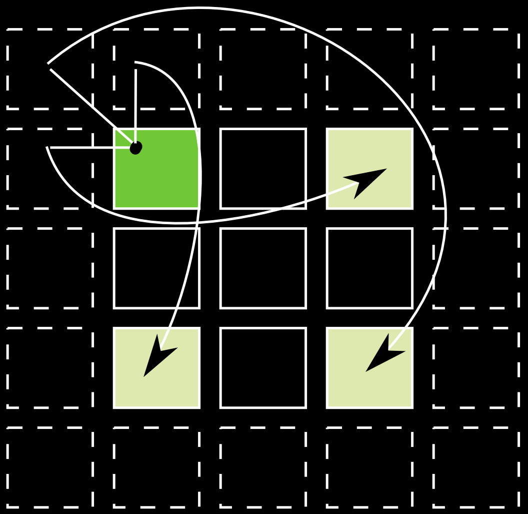
Send to only specific neighbors, indicated in various ways



Send to all neighbors



Send to all neighbors near enough to a target point



Support for wraparound neighbors (periodic boundary conditions)

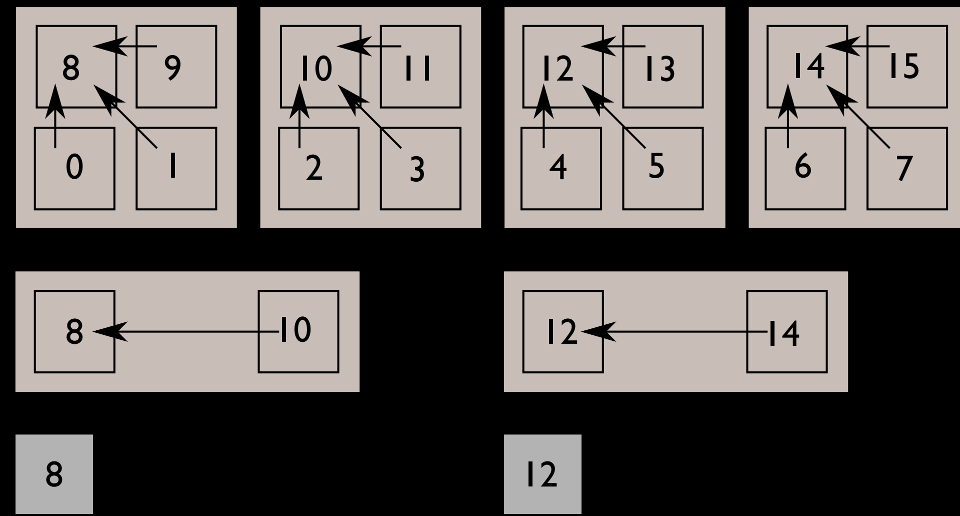
Global Communication Patterns

Merge-reduce

Round 0
k = 4

Round 1
k = 2

Results

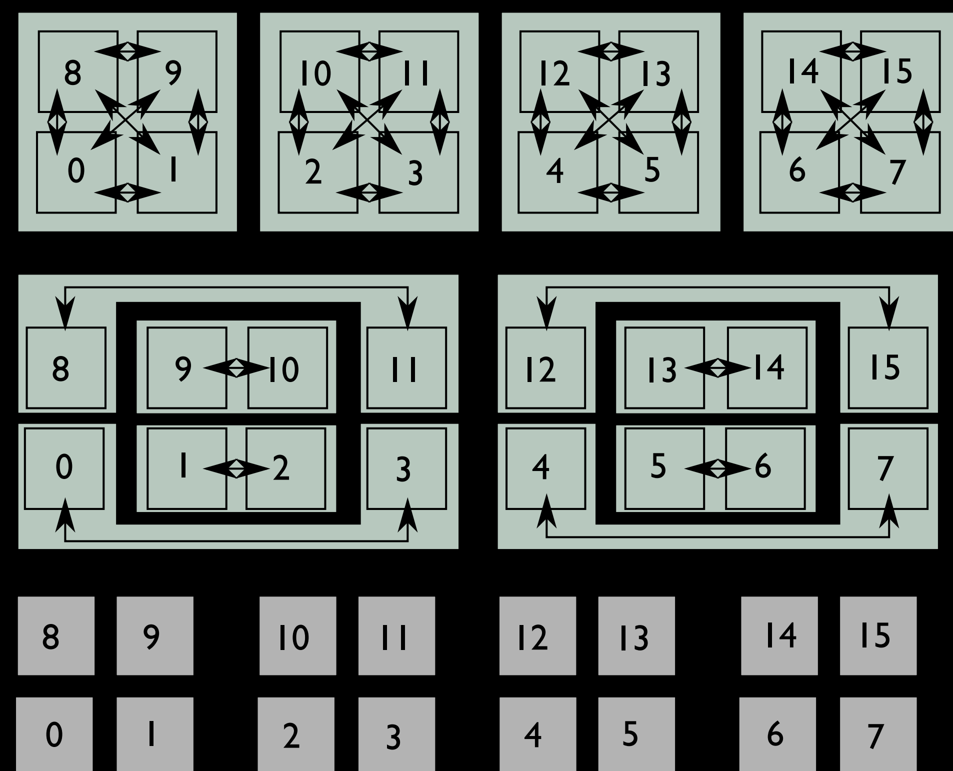


Swap-reduce

Round 0
k = 4

Round 1
k = 2

Results



Example Usage

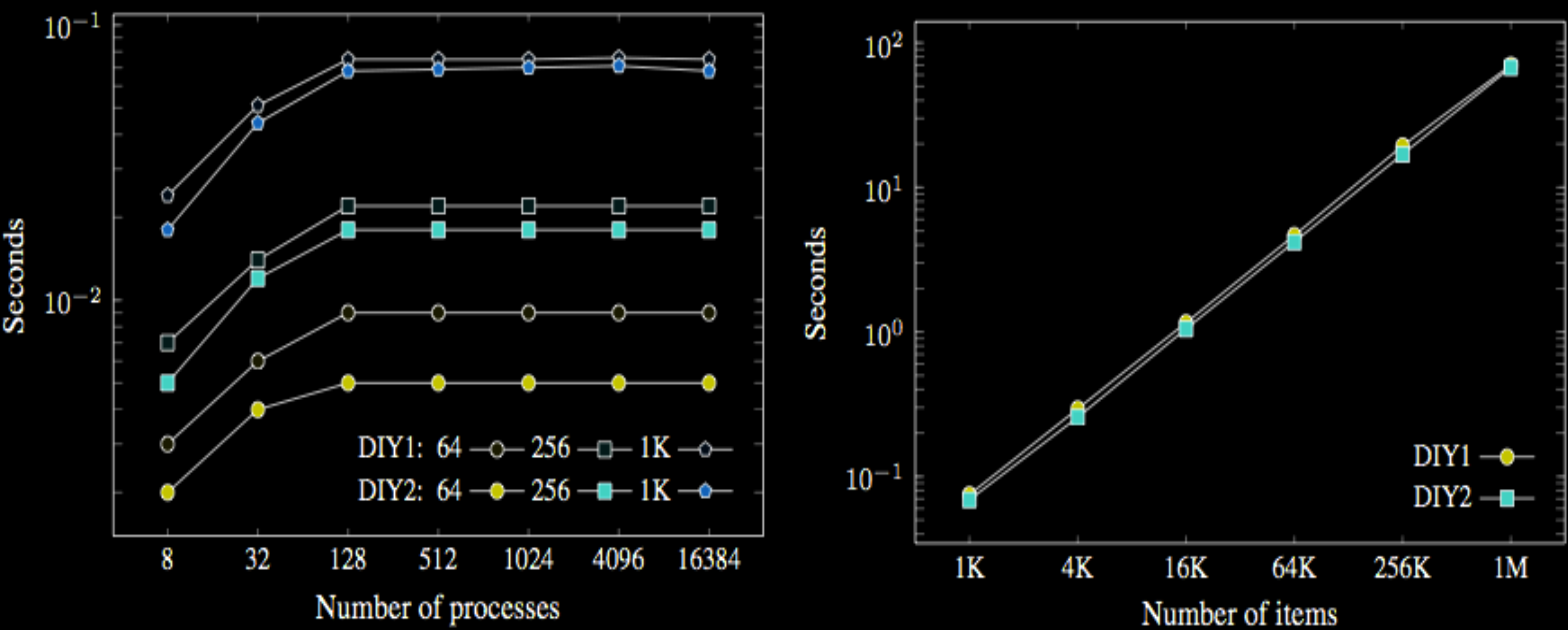
```
// initialization
Master                      master(world, num_threads, mem_blocks, ...);
ContiguousAssigner     assigner(world.size(), tot_blocks);
decompose(dim, world.rank(), domain, assigner, master);

// compute, neighbor exchange
master.foreach(&foo);
master.exchange();

// reduction
RegularSwapPartners(dim, tot_blocks, k);
reduce(master, assigner, partners, &foo);

// callback function for each block
void foo(void* b, const Proxy& cp, void* aux)
{
    for (size_t i = 0; i < in.size(); i++)
        cp.dequeue(cp.link()->target(i), incoming_data);
    // do work on incoming data
    for (size_t i = 0; i < out.size(); i++)
        cp.enqueue(cp.link()->target(i), outgoing_data[i]);
}
```

Performance Matters

Benchmark Results and Full Applications

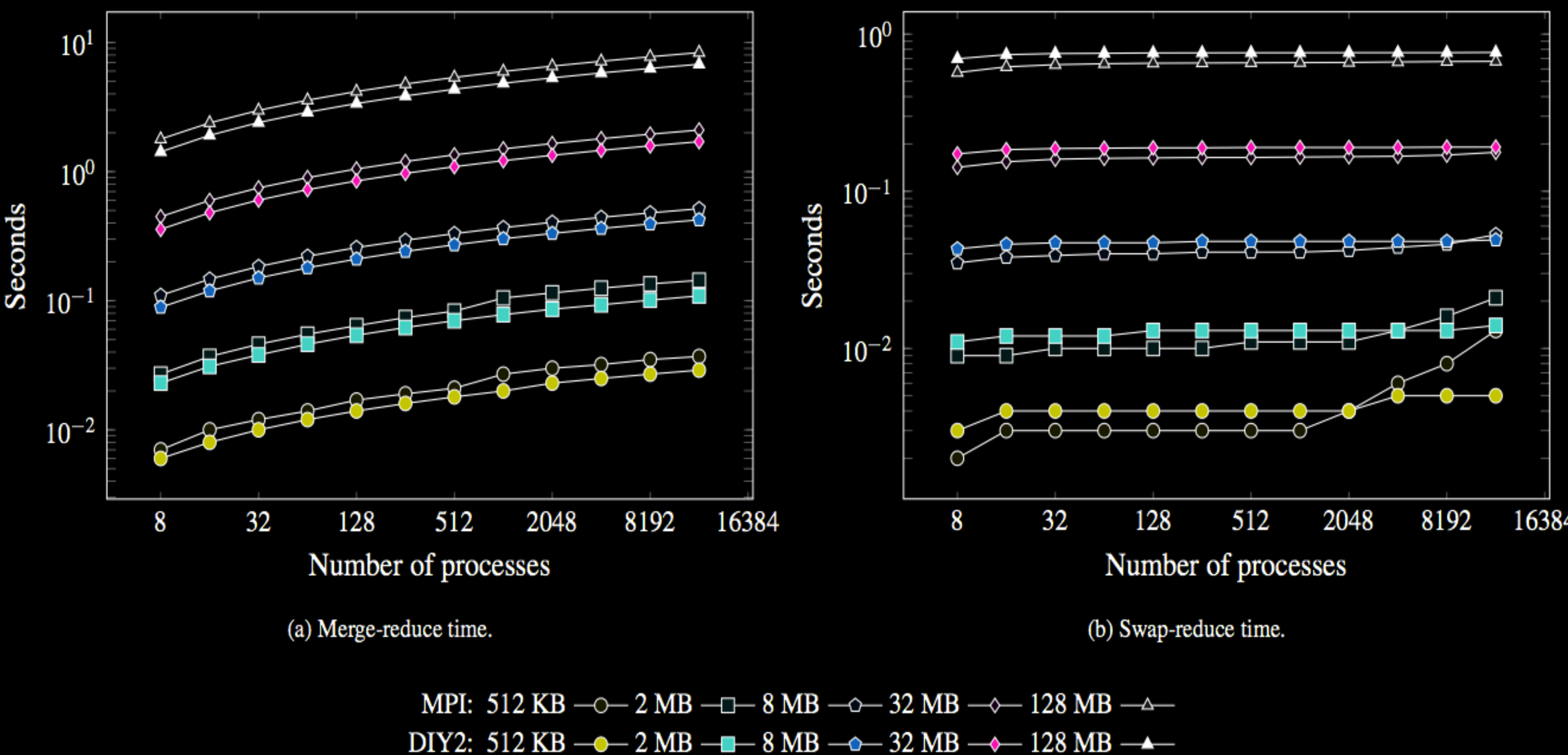
Neighbor Exchange Benchmark

We stress tested our neighbor exchange algorithm for a large number of small (20-byte) items exchanged.



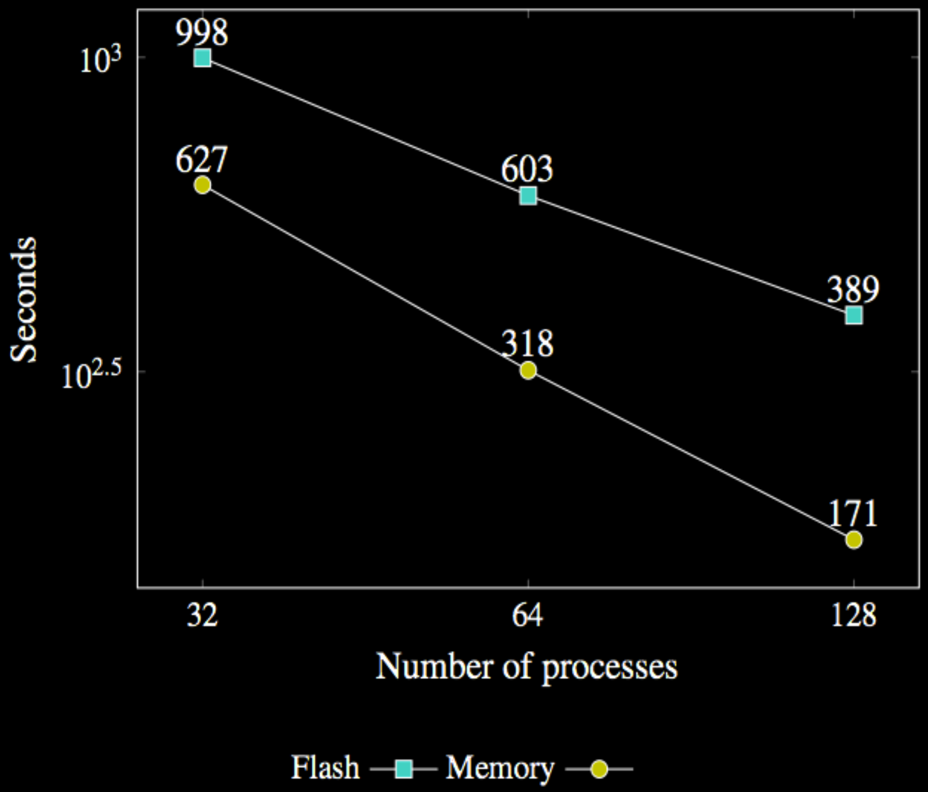
Conclusion: Linear complexity with total data size even though the data are divided into many small items. The user does not need to worry about aggregating data.

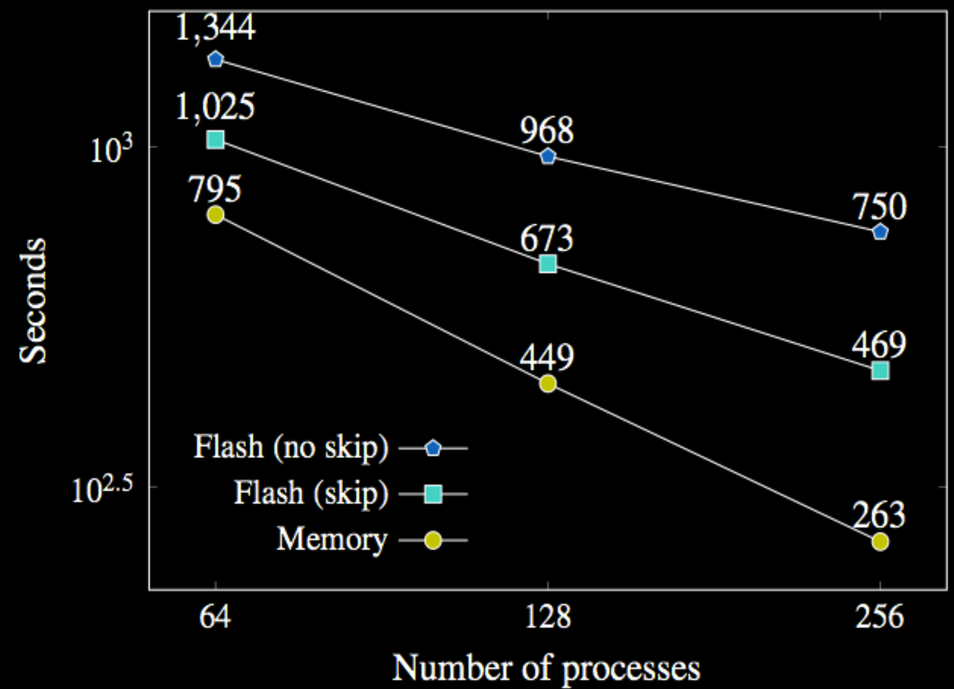
Global Reduction Benchmarks



Communication time only for our merge algorithm compared with MPI's reduction algorithm (left) and our swap algorithm compared with MPI's reduce-scatter algorithm (right).

Automatic Out-of-Core Algorithms



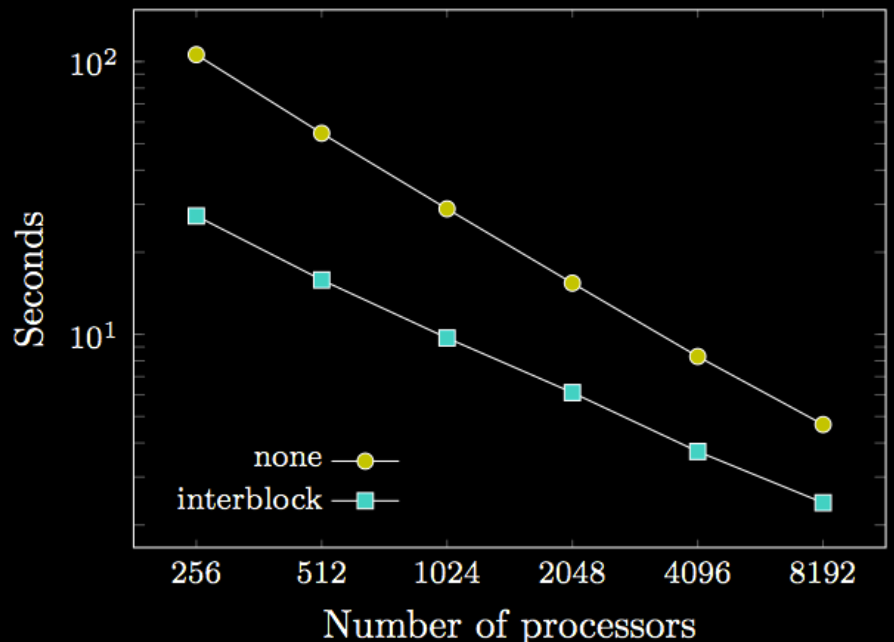


In- and out-of core performance of Delaunay tessellation.

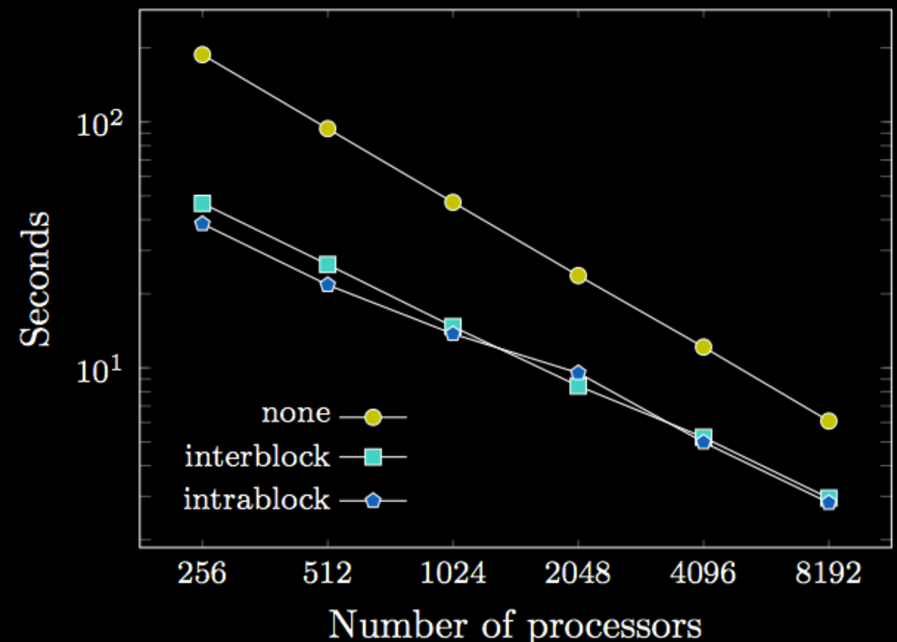
In- and out-of-core performance of distance field computation for watershed segmentation.

No source code changes required to switch between in-core and out-of-core.

Automatic Multithreaded Algorithms



(a) tess



(b) dense

Automatic threading of Voronoi tessellation.

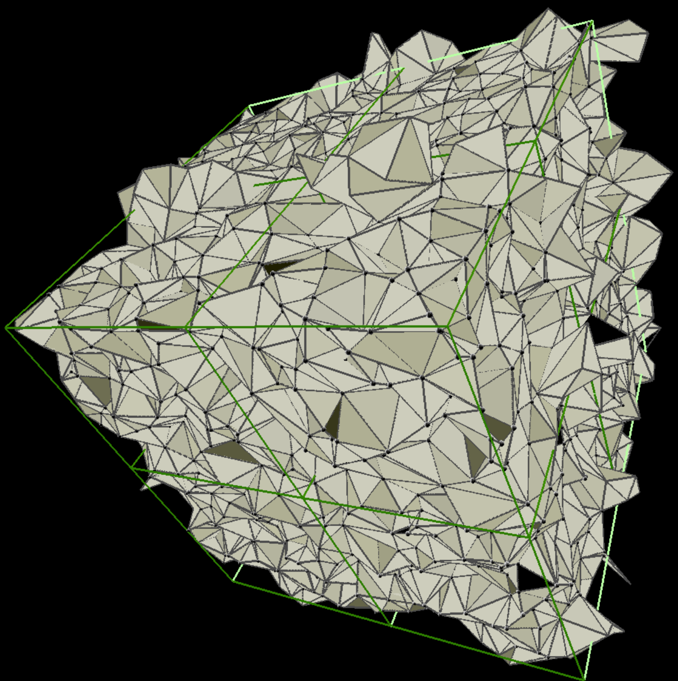
Comparison between manual and automatic threading of density estimation.

No source code changes required to switch between single and multithreaded.

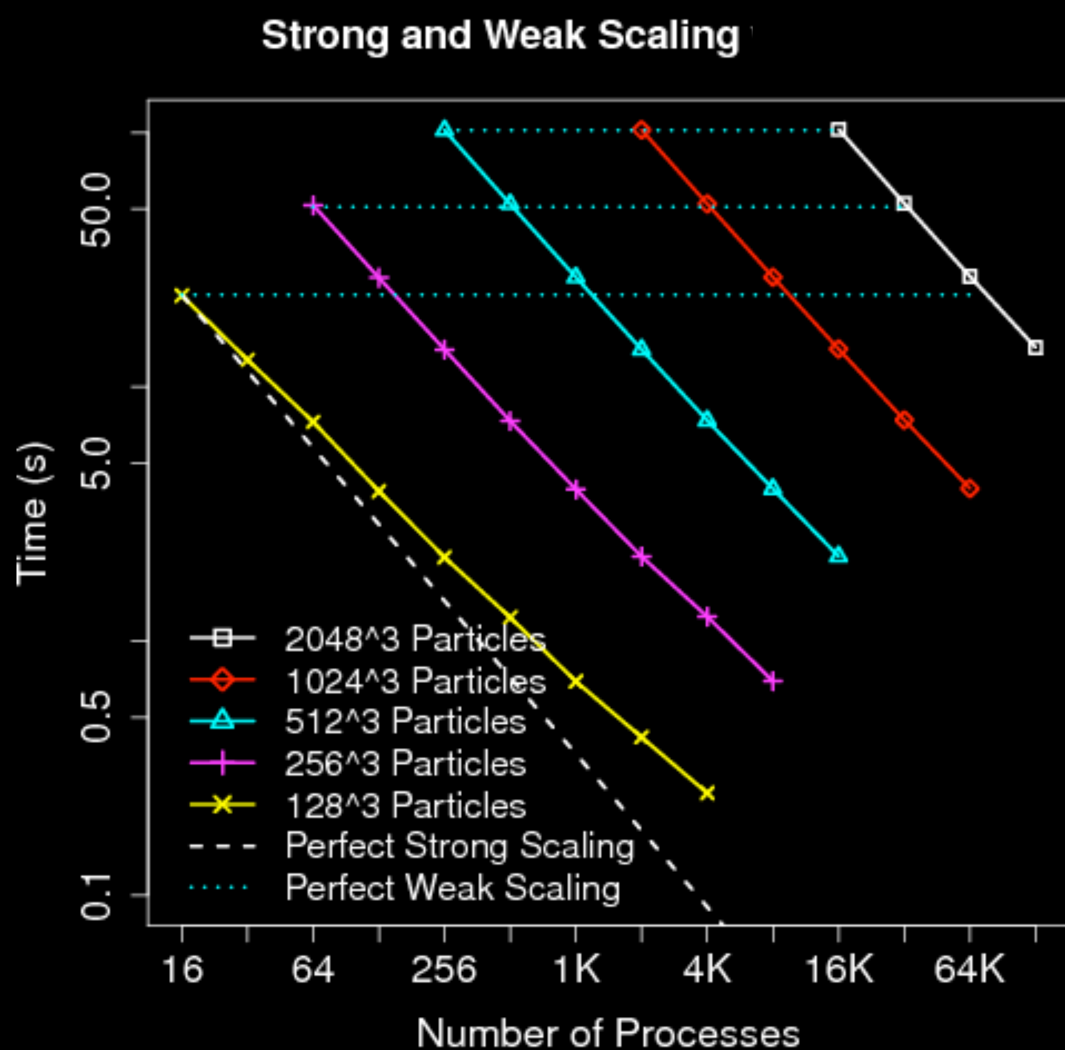
With Dmitriy Morozov and Carolyn Phillips

[github.com/diatomic/tess2]

Computational Geometry in Cosmology



Strong and weak scaling for up to 2048^3 synthetic particles and up to 128K processes (excluding I/O) shows up to 90% strong scaling and up to 98% weak scaling.

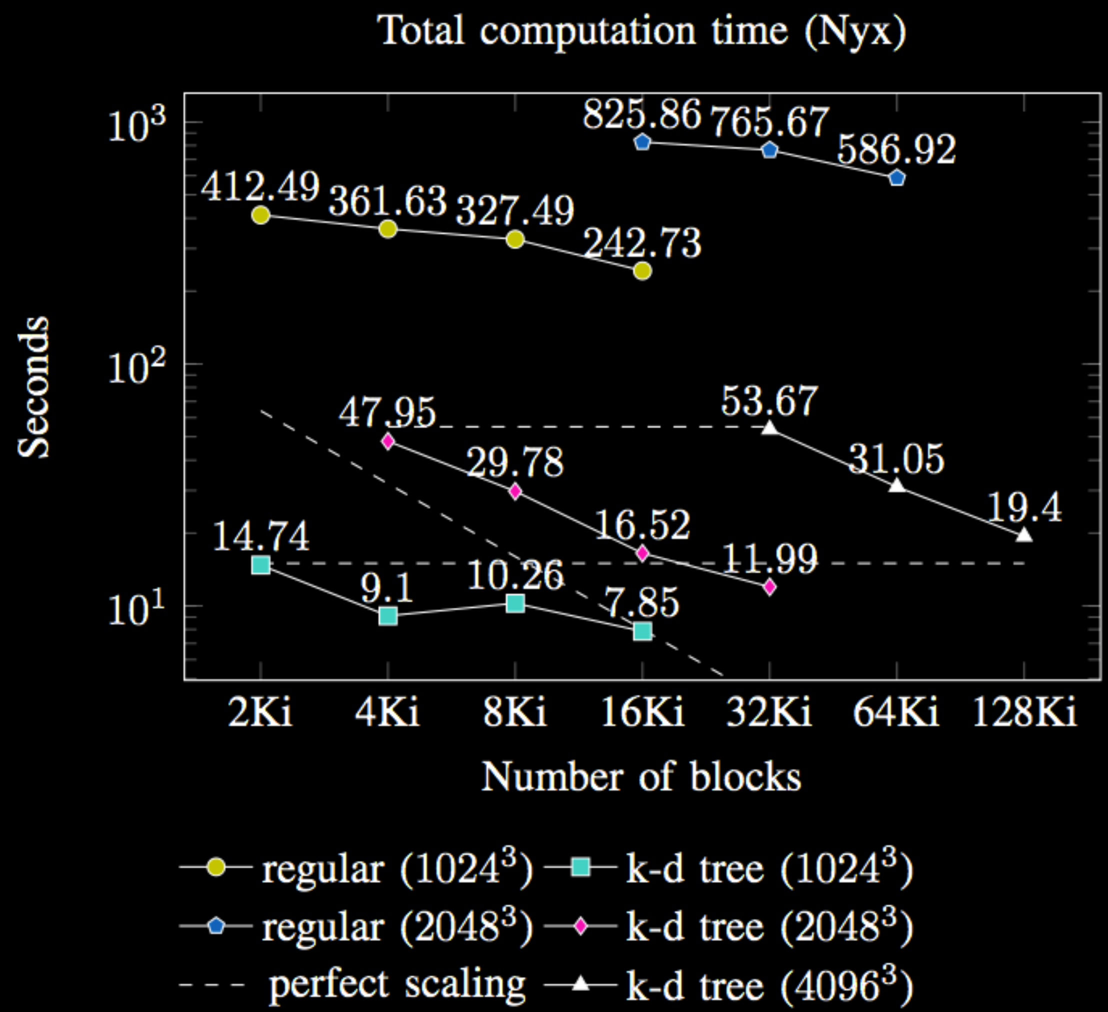


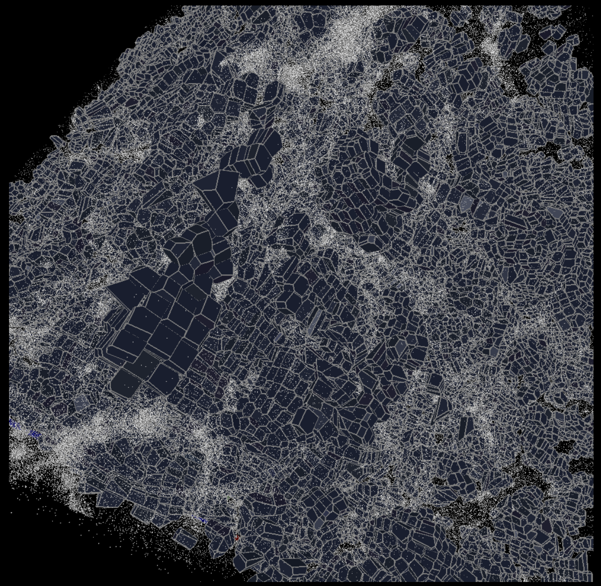
Peterka et al., High-Performance Computation of Distributed-Memory Parallel 3D Voronoi and Delaunay Tessellation. SC14.

Courtesy Dmitriy Morozov

[github.com/diatomic/tess2]

Load Balancing in Cosmology





Cosmology simulations have severe load imbalance. Tessellating meshes using a k-d tree instead of regular grid results in dramatically improved performance.

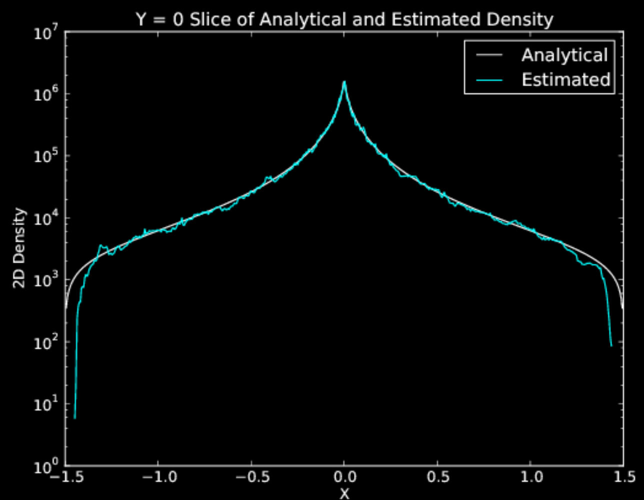
Morozov and Peterka, Efficient Delaunay Tessellation Through K-D Tree Decomposition, to appear SC16.

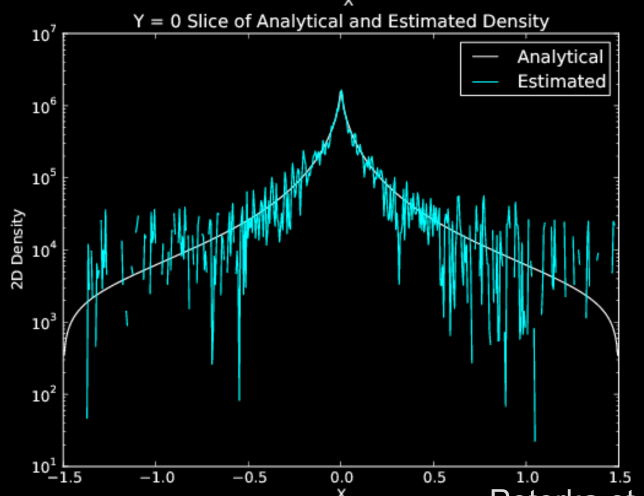
With Hadrien Croubois, Nan Li, Steve Rangel, and Franck Cappello

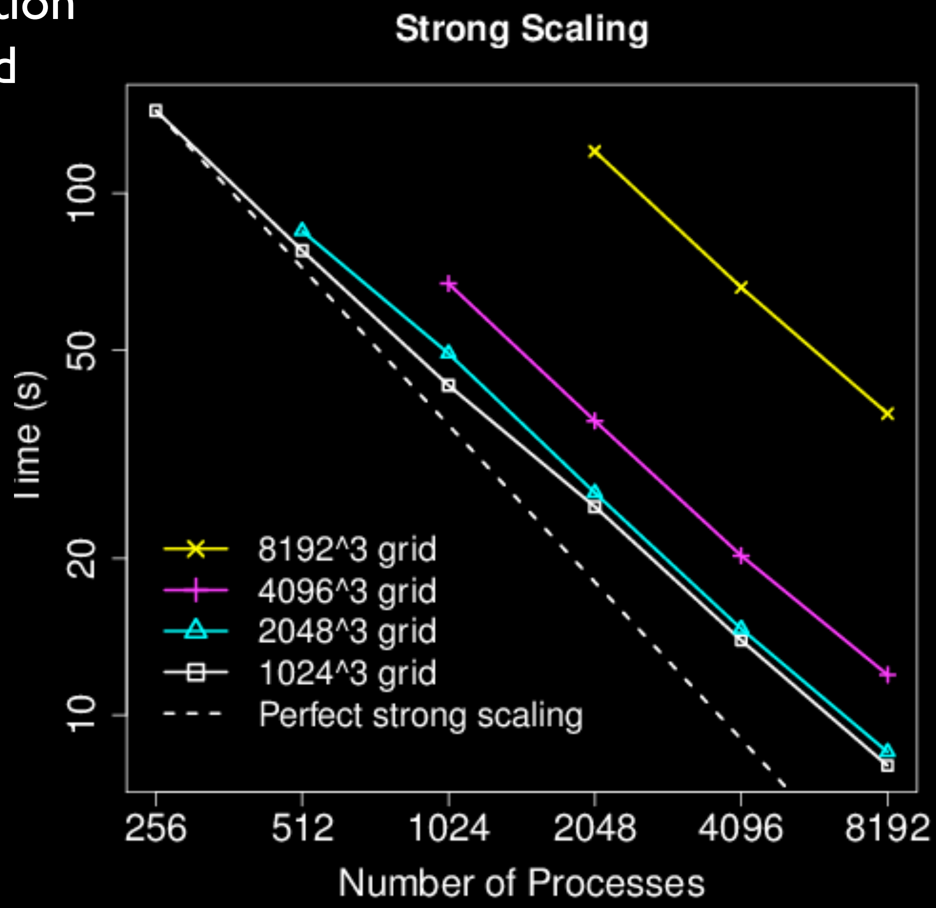
[github.com/diatomic/tess2]

Density Estimation in Cosmology

Tessellation-based density estimation is parameter free, shape free, and automatically adaptive







Above: Strong scaling of estimating the density of 512^3 synthetic particles onto grids of various sizes.

Left: comparison of tessellation-based and CIC density

Peterka et al., Self-Adaptive Density Estimation, SIAM SISC 2016.

Recap

Block Parallelism

Block abstraction for parallelizing data analysis allows one to:

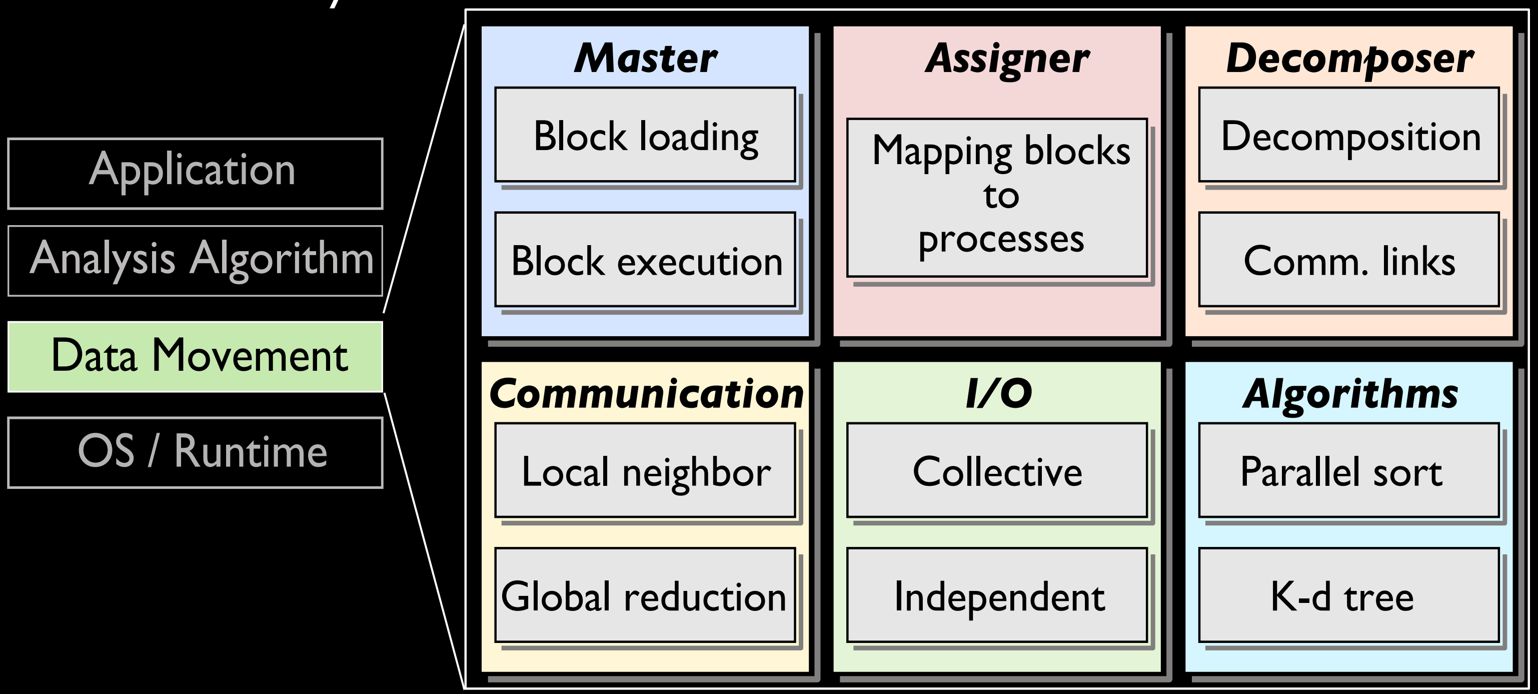
- Decompose data into blocks
- Assign blocks to processing elements
- Have several decompositions at once
- Overload blocks, migrate blocks between processing elements
- Communicate between blocks
- Migrate blocks in and out of core
- Thread blocks with finer-grained processing elements

All made possible by choosing blocks as the parallel abstraction

Think Blocks!

Tom Peterka, ANL
Dmitriy Morozov, LBNL
github.com/diatomic/diy2

Software: DIY



DIY is a programming model and runtime for HPC block-parallel data analytics.

- Block parallelism
- Flexible domain decomposition and assignment to resources
- Efficient reusable communication patterns
- Automatic dual in- and out-of-core execution
- Automatic block threading

References

DIY Papers

- Peterka, Ross, Kendall, Gyulassy, Pascucci, Shen, Lee, Chaudhuri: Scalable Parallel Building Blocks for Custom Data Analysis. LDAV 2011.
- Peterka, Ross: Versatile Communication Algorithms for Data Analysis. EuroMPI 2012.
- Morozov, Peterka: Block-Parallel Data Analysis with DIY2. Submitted to LDAV 2016.

Selected DIY Application Papers

- Morozov, Peterka: Efficient Delaunay Tessellation through K-D Tree Decomposition. To appear SC16.
- Peterka, Croubois, Li, Rangel, Cappello: Self-Adaptive Density Estimation of Particle Data. SIAM Journal on Scientific Computing SISC Special Section on CSE 2015.
- Peterka, Morozov, Phillips: High-Performance Computation of Distributed-Memory Parallel 3D Voronoi and Delaunay Tessellation. SC14.
- Lu, Shen, Peterka: Scalable Computation of Stream Surfaces on Large Scale Vector Fields. SC14.
- Nashed, Vine, Peterka, Deng, Ross, Jacobsen: Parallel Ptychographic Reconstruction. Optics Express 2014.
- Gyulassy, Peterka, Pascucci, Ross: The Parallel Computation of Morse-Smale Complexes. IPDPS 2012.
- Nouanesengsy, Lee, Lu, Shen, Peterka: Parallel Particle Advection and FTLE Computation for Time-Varying Flow Fields. SC12.
- Chaudhuri, A., Lee-T.-Y., Zhou, B., Wang, C., Xu, T., Shen, H.-W., Peterka, T., Chiang, Y.-J.: Scalable Computation of Distributions from Large Scale Data Sets. LDAV 2012.

Acknowledgments

Facilities
Argonne Leadership Computing Facility (ALCF)
Oak Ridge National Center for Computational Sciences (NCCS)
National Energy Research Scientific Computing Center (NERSC)

Funding
DOE SDMAV Exascale Initiative
DOE SciDAC SDAV Institute

People
Dmitriy Morozov (LBNL)

https://github.com/diatomic/diy2

Tom Peterka
tpeterka@mcs.anl.gov
http://www.mcs.anl.gov/~tpeterka
Mathematics and Computer Science Division